

SRL-0058-TM

②

AR-006-440

AD-A246 435



DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
SURVEILLANCE RESEARCH LABORATORY
SALISBURY, SOUTH AUSTRALIA

TECHNICAL MEMORANDUM

SRL-0058-TM

BUFFERED SERIAL DATA CARD

DTIC
ELECTE
FEB 26 1992
S D

G. FIELKE

Distribution: Approved for Public Release

This document has been approved
for public release and sale; its
distribution is unlimited.

© COMMONWEALTH OF AUSTRALIA

COPY No. 2

MAY 1991

92 2 20 045

92-04508
■■■■■■■■■■

DEPARTMENT OF DEFENCE
 DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
 SURVEILLANCE RESEARCH LABORATORY
 SALISBURY, SOUTH AUSTRALIA

TECHNICAL MEMORANDUM

SRL-0058-TM

BUFFERED SERIAL DATA CARD

G. FIELKE

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Availability for Special
A-1	

SUMMARY

NAVSTAR Global Positioning System (GPS) User Equipments (UEs) send out navigation information in the form of blocks of serial data. The format and transmitting rate of the data varies between UEs.

Several agencies within the Department of Defence require the collection of GPS UE data.

As a personal computer general purpose data logging card was not available commercially, it was decided to design and develop such a card to enable collection of data from any GPS UE.

The card may be used to collect data from any source which transmits data at either RS232 or RS422 levels.

This manual describes the operation of the Buffered Serial Data Card and the process of writing user software.



© COMMONWEALTH OF AUSTRALIA

POSTAL ADDRESS: Director, Surveillance Research Laboratory,
 PO Box 1650, Salisbury, South Australia 5108

TABLE OF CONTENTS

	Page No
1. INTRODUCTION	1
2. ADDRESSING THE BUFFERED SERIAL DATA CARD	1
3. FUNCTIONAL DESCRIPTION	2
4. OPERATING SOFTWARE	4
4.1 Initialising the Card	4
4.1.1 Using Turbo Pascal	5
4.1.2 Using a language other than Turbo Pascal	7
4.2 Writing a Main Program	10
4.3 Sample Main Program	12
5. CONCLUSION	14
ACKNOWLEDGMENT	14
REFERENCES	15

LIST OF APPENDICES

I. SAMPLE MAIN PROGRAM LISTING	16
--	----

LIST OF TABLES

1. ADDRESSING OFFSETS AND THE FUNCTIONS PERFORMED	23
---	----

LIST OF FIGURES

1. Circuit Schematic	24
2. Functional Units Block Diagram	25
3. Flowchart of Initialisation Procedure	26
4. Flowchart of an Operating Method	27

1. INTRODUCTION

NAVSTAR Global Positioning System (GPS) User Equipments (UEs) send out navigation information in the form of blocks of serial data. The format of the data, the transmitting baud rate, the block size and the transmitting rate of the blocks all vary between UEs.

Several agencies within the Department of Defence require the collection of GPS UE data.

A personal computer (PC) can perform this function as well as displaying and processing the data as it is being collected. Manufacturers of GPS equipment provide software to enable the collection of data generated by their particular equipment. A special card is generally required with the software which is suitable for an IBM PC/AT or compatible.

A general purpose card was not commercially available.

It was decided to design and develop a card to enable the collection of data from any GPS UE which provided either an RS232 or RS422 data port. The card is designed to transmit and receive serial data at various rates up to 76.8 kilo baud.

To facilitate the various serial data formats among GPS UEs, the card can be programmed to receive or transmit any desired format. This can be done, using the software provided, via a series of set up options which are menu driven. A 4k byte data buffer is used to temporarily collect the incoming data prior to being retrieved and stored by the PC.

The card and its associated software, although designed for use with GPS UEs, can be used in any situation where serial data must be collected and stored. It is a full length, switch addressable, 8 bit and IBM PC/XT/AT compatible card.

2. ADDRESSING THE BUFFERED SERIAL DATA CARD

The card is switch addressable within the 10 bit PC I/O addressing range, in 16 byte increments.

The base address is set by the DIP switch on the card and may be set to any 16 bit boundary. In order not to clash with any resident board in a PC, it is recommended that the prototype address area (Hex 300-31F) be used. Therefore, card addresses of Hex 300 and Hex 310 may be used. The card occupies 16 consecutive address locations beginning at the base address.

The card is initially set at the base address Hex 310. If the address space Hex 310-31F is not free, the base address must be changed by altering the DIP switch setting.

DIP switch number 1 must be open while switch number 8 is not used (see circuit diagram shown in Figure 1).

SRL-0058-TM

Switch number 1 enables or disables the card. Therefore, if this switch is closed, the card is disengaged and cannot be addressed.

The following table indicates the switch positions which allow the card to be addressed at Hex 300 and Hex 310 where :

O (Open) = 1								
C (Closed) = 0								
Address	A9	A8	A7	A6	A5	A4		
Switch No.	1	2	3	4	5	6	7	8
Hex								
300	O	O	O	C	C	C	C	X
310	O	O	O	C	C	C	O	X

Note that the 4 least significant address bits (A3 - A0) of the base address are not set by the DIP switch. These bits are used by the operating software to select and enable the card to perform specific functions. These bit values, and the corresponding function that is executed, are outlined and explained in the following section.

3. FUNCTIONAL DESCRIPTION

The circuit diagram for the Buffered Serial Data Card is shown in Figure 1. The operation of the card can be clarified by reducing the circuitry to the main functional units as shown in the block diagram of Figure 2.

These units are listed and briefly described below :

- a) Baud Rate Generator
- b) I/O Level Converter
- c) DART : Dual Asynchronous Receiver/Transmitter
- d) Dual Retriggerable Monostable Module
- e) Read/Write - Data, Addressing Module
- f) FIFO : First In/First Out Buffer

3.1 Baud Rate Generator

A crystal oscillator circuit and a latch are used to generate the receive and transmit clock frequencies for the DART. The oscillator frequency is divided by 5 and sent to a 7 stage binary counter to produce 7 discrete clock frequencies which are in turn connected to the input pins of two 8 line to single line multiplexers. The multiplexers are separately addressed; one to select the receiving data baud rate, and the other to select the transmitting data baud rate.

The transmitting and receiving baud rates are selected through software, as explained in Operating Software.

3.2 I/O Level Converter

This unit consists of RS232 and RS422 receivers and drivers, and a latch which is used to select the desired interface. The desired interface is software selectable by the user.

3.3 DART : Dual Asynchronous Receiver/Transmitter

The DART is used as a serial-to-parallel converter to receive data, and a parallel-to-serial converter to transmit data. The DART programmable options include 1, 1.5, 2 stop bits, even, odd or no parity, and 5, 6, 7 or 8 bit receive (Rx) or transmit (Tx) word lengths. It is used in the 16 times clock mode ie. the transmitter and receiver can handle data at a rate of 1/16 of the clock rate supplied to the Tx and Rx clock inputs.

The clock on the PC bus - pin B20 of the expansion slot - is divided by 2, buffered and used as the system clock for the DART. The PC bus clock should be no more than 12MHz - twice the specified 6MHz system clock frequency of the DART.

The DART has three 8 bit receive buffer registers and outputs a pulse on the Wait/Ready line to indicate that data has been received.

3.4 Dual Retriggerable Monostable Module

When triggered, this module provides two pulses. One to read a data byte from the DART, and another to write the byte to the FIFO buffer. This process, once enabled by software, is hardware automated so that when data enters the card, a pulse from the DART triggers this module to send the 2 pulses which transfer the data directly to the First In/First Out (FIFO) buffer. The pulse widths are set by resistor/capacitor combinations.

The process can be enabled and disabled through the user software via a start latch.

3.5 Read/Write - Data, Addressing Module

This module contains the interface logic between the card and the PC. The data bus interface is bi-directional so that data on the PC data bus can be written to the data bus of the card and vice versa.

Addressing the card in software serves to select the card (by using the base address - see Addressing the Buffered Serial Data Card), and to enable the card to perform a function by using the base address plus a 4 bit offset.

The address lines A9-A4 of the PC are equated with the DIP switch setting to enable the card. The least 3 significant bits of the offset, address lines A2-A0, are connected to the select pins of a 3-to-8 line decoder. The output lines of the decoder, along with the input/output read and write lines from the PC, are connected to specific parts of the card circuitry, such as latches and IC reset pins. When a decoder output line is set active a particular function will be executed.

Bit 4 of the address, A3, is sent to the DART to select the type of information (control or data) being transferred on the data bus.

The offset and the corresponding function the card performs, is shown in Table 1.

SRL-0058-TM

3.6 FIFO : First In/First Out Buffer

This circuit loads and releases data on a first in, first out basis. Hence, no addressing is required. The FIFO buffer has a capacity of 4k bytes. Data is toggled in and out by the use of its WRITE and READ pins.

The status of the buffer can be determined by inspecting, through software, two flags which indicate :

- a) if data is present in the buffer (empty flag), and
- b) if the buffer is full and an overflow has occurred (full flag).

The full flag is externally latched.

4. OPERATING SOFTWARE

If Turbo Pascal is unavailable, the software to operate the card may be written by the user in another language. A sample main program has been written, using Turbo Pascal Version 4, to successfully initialise and run the card. The program listing is provided in Appendix I, and may be used as a guide for the user in writing a customised program.

The object code of this program can be supplied upon request.

The sample main program uses two Turbo Pascal Units (TPUs).
These units are :

- a) 'menuinit.TPU' - used to initialise the card by leading the user through a series of menus. The flowchart shown in Figure 3 illustrates the initialisation procedure.
- b) 'std_cons.TPU' - contains a list of the names of the standard constants used by both 'menuinit.TPU' and the sample main program. The names and their values contained in this unit are shown in the listing of the sample main program. These names are used within the main program to improve its readability.

The object code of these TPU files is available to assist in the initialisation procedure.

The following describes how these units may be used in conjunction with a Turbo Pascal main program, and how to initialise and run the card using another language.

4.1 Initialising the Card

To initialise the card, control bytes which indicate the format of the serial data to be transmitted or received must be written to the DART circuit on the card. The interface type and the serial data baud rates must also be selected.

4.1.1 Using Turbo Pascal

The TPU files are provided to initialise the card from a main program if a Turbo Pascal compiler is available.

This is accomplished by using, in the main program, the following procedure and parameters :

```
list_set(www,aaa)
```

where

```
list_set  -   is a procedure in 'menuinit.TPU' which
               displays the first menu and then passes
               control to other procedures within
               'menuinit.TPU' to determine values for the
               control variables which are used to set up
               the DART chip.

www       -   is a byte which is returned to the main
               program by 'list_set' and corresponds to the
               word length of the serial data to be
               received. This byte is used to mask the
               data bytes from the FIFO buffer.

aaa       -   is the base address word of the card as set
               by the DIP switch (eg. $310). This word is
               sent to 'list_set' and used within
               'menuinit.TPU' to send the control bytes to
               the DART.
```

List_set is declared in 'menuinit.TPU' as

```
PROCEDURE list_set( VAR www : BYTE;
                   aaa : WORD );
```

It was anticipated that the main use of this card would be in the logging of data from Global Positioning System (GPS) user equipments and hence the first menu in 'list_set' lists the possible GPS receivers that may be used :

```
Which GPS receiver is being used?
*****
```

```
Magnavox MX4400           1
Raytheon Raystar 920       2
Rockwell Collins 3A       3
Texas Instruments TI4100   4
Trimble Trimpack          5

None of the above         0
```

Other GPS receivers may be added later as the output data format is known.

SRL-0058-TM

If the GPS receiver being used is not included in this menu, or a data source other than a GPS receiver is being used, the user must know the format of the serial data to be logged. In this situation, the user chooses 'None of the above' and is led through the initialisation by a series of menus.

The appropriate serial data parameters are selected from the following list of format options :

```

Interface : RS232
           RS422

Data Source Transmitting Baud Rate : 76.8k , 38.4k
                                     19.2k , 9.6k
                                     4.8k , 2.4k
                                     1.2k , 600

Data Source Receiving Baud Rate   : 76.8k , 38.4k
                                     19.2k , 9.6k
                                     4.8k , 2.4k
                                     1.2k , 600

Word Length : 5, 6, 7, 8 bits/chr

Parity : None
        Odd
        Even

Stop Bits : 1, 1.5, 2

```

After each selection is made, a control byte is assigned to a variable. After the last menu selection, the bytes are sent to the DART circuit to be initialised.

If a GPS receiver is selected from the first menu, a list of standard preset parameters for that particular receiver is displayed. The user is then prompted to verify these parameters. A typical example is shown below.

Rockwell Collins 3A

```

Interface      : RS422
Transmit Baud Rate : 76.8k
Receive Baud Rate : 19.2k
Word Length    : 8 bits/chr
Parity         : Odd
Stop Bits      : 1

```

Are these correct? (y/n) =>

If these are accepted, the card is initialised.

If not, the user is led through each parameter menu, as before, to select the appropriate data format before the card is initialised.

When this has been completed, a byte is passed back to the main program. This byte corresponds to the word length of the serial data (as selected from the menus).

ie.	Word Length (bits/chr)	Passed Byte
	8	\$FF : 1111 1111
	7	\$7F : 0111 1111
	6	\$3F : 0011 1111
	5	\$1F : 0001 1111

This byte is needed for word lengths of 5, 6 or 7 bits to mask the data byte from the FIFO to obtain the original data. The data bus from the DART to the FIFO is tied high. This forces the unused bits of a data byte (ie. D6, D7 for 6 bits/chr) to be set to '1'. These unused bits must be reset to '0' before storing the data byte on disk. This is accomplished by ANDing the data byte with the corresponding word length byte shown above.

4.1.2 Using a language other than Turbo Pascal

The following explains how to initialise the card if a Turbo Pascal compiler is unavailable or another language is preferred.

a) Setting the Transmit and Receive Baud Rates

To select a particular baud rate, a data byte is addressed to the location [base address + 1]. The four MSBs of the data byte, D4 - D7, select the card transmitting (Tx) data baud rate, while the four LSBs, D0 - D3, select the card receiving (Rx) data baud rate, as shown in the table below.

D7	D6	D5	D4	Tx Baud Rate	Rx Baud Rate
D3	D2	D1	D0		
X	0	0	0	76.8k	76.8k
X	0	0	1	38.4k	38.4k
X	0	1	0	19.2k	19.2k
X	0	1	1	9.6k	9.6k
X	1	0	0	4.8k	4.8k
X	1	0	1	2.4k	2.4k
X	1	1	0	1.2k	1.2k
X	1	1	1	600	600

The address, and hence the desired Rx and Tx baud rates, is latched until programmed differently.

SRL-0058-TM

b) Setting the Interface RS232/RS422

This selection is made by placing a '0' or '1' on the data bus while addressing the location [base address + 2].

D0 = 0 selects RS232
D0 = 1 selects RS422

This selection is latched until programmed differently.

c) Setting the Serial Data Format (see Reference 1)

The Z-80 DART circuit contains six write registers (WR0 - WR5) and three read registers (RR0 - RR2). The write registers are programmed separately by the user to configure the DART to receive or transmit a particular serial data format.

Programming the write registers requires writing a byte to WR0 to select the required register. Hence, with the exception of WR0, programming the write registers requires two bytes. The first byte selects the appropriate register; the second byte is the actual control word that is sent to the register. These control bytes are addressed to location [base address + 8].

The control bytes for the first two registers are :

WRITE REGISTER : CONTROL BYTE

WR0 : S18
WR1 : SE0

WR2 is not relevant to the operation of the card and is not used.

The control bytes sent to registers WR3 - WR5, depend on the serial data format. The bytes can be determined from the following bit assignments :

WRITE REGISTER 3

D7 D6 0 0 0 0 0 1

where

D7	D6	
0	0	: Rx 5 bits/char
0	1	: Rx 7 bits/char
1	0	: Rx 6 bits/char
1	1	: Rx 8 bits/char

WRITE REGISTER 4

0 1 0 0 D3 D2 D1 D0

where

<u>D1</u>	<u>D0</u>	
X	0	: No parity
0	1	: Odd parity
1	1	: Even parity

<u>D3</u>	<u>D2</u>	
0	0	: Not used
0	1	: 1 stop bit
1	0	: 1.5 stop bits
1	1	: 2 stop bits

WRITE REGISTER 5

0 D6 D5 0 1 0 0 0

where

<u>D6</u>	<u>D5</u>	
0	0	: Tx 5 bits/char
0	1	: Tx 7 bits/char
1	0	: Tx 6 bits/char
1	1	: Tx 8 bits/char

See Reference 1 for more detail.

Example : Configure the card to receive RS232 data, of the following format, at 9600 baud and transmit at 38.4k baud :-

Data format - 8 bits/chr
 Odd parity
 1 stop bit

The base address of the card is set by the DIP switch to Hex 310.

The following bytes would be sent to the address locations shown.

Address	Byte	
Hex 316	0	: to reset the DART before initialising. The byte has no relevance during reset.
Hex 312	0	: to select RS232.
Hex 311	Hex 13	: to select Rx baud rate of 9600 and a Tx baud rate of 38.4k.

SRL-0058-TM

Hex 318	Hex 18	:	control byte for WR0 to reset the channel (Note : no pointer required).
Hex 318	1	:	stored in WR0 to select WR1.
Hex 318	Hex E0	:	control byte for WR1.
Hex 318	3	:	select WR3.
Hex 318	Hex C1	:	control byte for WR3.
Hex 318	4	:	select WR4.
Hex 318	Hex 45	:	control byte for WR4.
Hex 318	5	:	select WR5.
Hex 318	Hex 68	:	control byte for WR5.

4.2 Writing a Main Program

Once the card has been initialised, the user may select software functions to operate the card. The functions available and their description (see also Table 1), are outlined as follows ;

- a) enabling/disabling the process of writing incoming data directly to the FIFO buffer.

This involves enabling/disabling the Dual Retriggerable Monostable module via a start latch.

Writing a '1' to address location [base address + 4] will set the latch to enable this process to occur.

This process may be halted, if desired, by writing a '0' to address location [base address + 4].

- b) inspecting the status of the FIFO buffer.

The full and empty flags of the FIFO buffer may be checked by reading a byte on the data bus at address location [base address + 3].

This byte will contain information in the second (D1) and third (D2) bits which reveal whether data exists in the FIFO, and whether the FIFO buffer has overflowed.

When D1 is high, the buffer contains data.

When D2 is high, the buffer has overflowed and data has been lost.

When these bits are low, the converse is the case.

The full flag is externally latched such that if the buffer overflowed momentarily, the D2 bit of the status byte would remain high, indicating that data had been lost. The latch remains set until reset by software.

- c) reading the data from the FIFO buffer.

When the buffer contains data, the first data byte is toggled out and onto the data bus, and is read by addressing the location [base address + 5].

- d) clearing the full flag latch.

This latch is cleared by writing a byte to address location [base address + 7]. The byte value is irrelevant.

- e) transmitting data from the card.

To transmit data from the card, the DART is first initialised as previously described. The data byte to be transmitted, bypasses the FIFO and enters the DART in parallel form. It is converted to a serial format and supplied with a start bit, parity and the selected number of stop bits before being transmitted from the card at the selected level - RS232/RS422.

The start latch must be disabled before transmitting data by writing a '0' to address location [base address + 4].

The status of the transmit buffer of the DART must be continually monitored to ensure error free transmission. This can be performed by reading the contents of the DART read register RR0 (see Reference 1). The third bit in this register (D2) indicates the required status. Writing a '0' to the pointer register WR0 (ie. to address location [base address + 8]) will enable a read of RR0 by assigning the control byte of address location [base address + 8] to a byte variable.

When the third bit of this byte is high ('1'), the transmitting buffer is empty and a byte can be transmitted through the DART.

The data is sent through the card when addressed to location [base address], ie. offset = 0.

- f) operation using interrupts.

The Buffered Serial Data Card is equipped with the facility to be interrupt driven for faster operation.

The empty flag of the FIFO is connected to a PC interrupt line via a jumper on the card. This jumper is used to select the desired Interrupt Request (IRQ) Level. Selection is made by placing the jumper across the appropriately labelled pair of pins.

SRL-0058-TM

When the FIFO contains data, the empty flag and hence the selected interrupt line go high.

The IRQ Level jumper is initially set to FLOAT to disable the interrupt facility.

4.3 Sample Main Program

The basic flowchart for the sample main program is shown in Figure 4. It illustrates the approach that may be taken to successfully operate the card.

This program uses a polled technique rather than interrupts and hence the IRQ Level jumper should be set to FLOAT.

The card is first initialised by entering the procedure 'list_set', as previously explained.

The program then prompts the user for a disk drive letter to store the data. The data files are assigned the names - RUN1.DAT, RUN2.DAT, RUN3.DAT etc. The data file name is prefixed by the disk drive letter.

eg. A:RUN1.DAT

The process of incoming data being read from the DART and written to the FIFO buffer is enabled.

However, the three 8 bit receive buffers of the DART will be occupied by unwanted information from earlier GPS data blocks. These three bytes will then be the first bytes to be transferred to the FIFO and stored to file. Therefore, this program reads the first three bytes from the FIFO but does not write them to the data file.

The message displayed on the screen during normal operation indicates

- 1) that the card is running,
- 2) the three options that may be chosen, and
- 3) the name of the data file.

The message is written to the screen once but never cleared during logging. Hence it is continually displayed. The number of data bytes collected is also displayed and updated.

The status of the FIFO buffer is continually monitored by polling.

When the FIFO empty flag indicates that the buffer contains data, a check is made on the status of the latched full flag of the buffer. If the flag has been set, data has been lost and a visual and temporary audible indication is given. Data logging continues but the visual indicator remains for the duration of logging to this particular data file, unless the user intervenes.

If the full flag has not been set, a data byte is read from the buffer, ANDed with the word length byte and stored in the data file. A count variable is incremented to indicate the number of bytes collected.

If no data exists in the FIFO, as may occur during the interval between successive incoming blocks of data, the number of bytes collected is indicated on the screen.

After each data byte is read and stored in the file, a check is made on the number of bytes stored to see if it has exceeded the preassigned limit of 340k. This limit is chosen to safely enable one data file per 360k capacity floppy disk.

The keyboard action to execute the three options available to the user during collection of data are :

- 1) 'q' : to close the current file and quit
- 2) 'n' : to close the current file and continue storing incoming data to the next file (with no loss of data)
- 3) 'r' : to close the current file and ignore incoming data until ready (this is a temporary suspension of logging which foregoes the need to reinitialise the card to continue)

If 340k has not been exceeded, a KEYPRESSED check is made (see Reference 2). If no key has been pressed, the program returns to reading the status byte. This sequence of events continues until a key is pressed or the data byte count reaches 340k.

When 340k bytes of data has been collected, program control is passed to the procedure 'logged340'. This procedure gives an audible and visual indication and prompts the user to either renew the floppy disk and continue logging to a new file or to quit data logging.

When a key is pressed, the character is read and inspected. If the character is 'q', 'n' or 'r' (or the uppercase equivalent) then the corresponding function is executed.

On termination of logging, a final message is displayed indicating the number of data bytes collected in the latest data file.

The FIFO buffer can store 4k bytes of data before overflowing and losing the proceeding data.

When the logging is suspended, either by entering 'n' during the normal process of logging data, or when 340k bytes of data has been stored, the next 4k bytes of incoming data will be collected in the FIFO ready to be read out and written to the next file.

If logging is resumed before the FIFO is full, no data will be lost.

As a guide, if data blocks of size X bytes are entering the card every second, data loss will be prevented if logging is continued within about $4000/X$ seconds after its suspension.

SRL-0058-TM

5. CONCLUSION

The Buffered Serial Data Card is able to collect data from any RS232 or RS422 data source. The card plugs in to any IBM or compatible machine and is simple to program in a suitable language.

The Buffered Serial Data Card has been used to collect data from a variety of GPS receivers.

The card has been used with a Rockwell Collins UH receiver to record data at 76.8k baud and issue commands to the receiver at 19.2k baud. The card has also recorded data from the following GPS receivers : Magnavox 4400, Raytheon Raystar 920, Magellan and Trimble Trimpack.

ACKNOWLEDGMENT

Acknowledgment is made to Mr. Mark Knight, of SRL DSTO, who undertook the hardware design phase of the Buffered Serial Data Card.

REFERENCES

No.

- 1 Zilog Components Data Book.
(Z8470 DART)
- 2 Turbo Pascal 4 Users Manual
BORLAND International

SRL-0058-TM

APPENDIX I

SAMPLE MAIN PROGRAM LISTING

PROGRAM SAMPLE;

{ Ver 3.8 }

```

{*****
  This program is used in conjunction with the plug-in buffered serial data
  card to log blocks of serial data onto floppy disk.

  The buffer card must first be initialised to enable it to accept the
  appropriate serial data configuration. Procedures which exist in the unit
  'menuinit.TPU' are used to prompt the user to input the data configuration
  ie. data source transmitting and receiving baud rates, word length, parity,
  stop bits and whether RS232 or RS422 is being used. The buffer card is then
  initialised to receive data in the particular configuration chosen.

  After initialisation, the user is then prompted to input the disk drive
  that the data is written to.
  The program assigns a limit of 340k on the size of a data file created.
  The data files are given the names RUN1.DAT, RUN2.DAT etc (prefixed by the
  disk drive letter).

  The program then enables the FIFO buffer to accept data. The number of
  bytes written to the data file is updated on the screen when there is no
  data in the FIFO.

  The logging can be continued to another file on the same drive without
  losing data or to another drive when ready (when logging is suspended,
  incoming data is ignored).

  When 340k bytes of data have been written, the logging is suspended and
  a change of floppy disks can be accomplished. The data file name is then
  incremented.

  The FIFO buffer has a capacity of 4k bytes, therefore when it is desirable
  to change file names, or when 340k bytes has been written, the next 4k
  bytes of data is saved. If the logging of data is not resumed before the
  buffer is full, the buffer will overflow and data will be lost. This is
  indicated by a message on the screen when the data logging is continued
  after its suspension. Hence, it is possible, if desired, to change disks
  and continue logging without losing data. If blocks of data of size X bytes
  are logged every second, the FIFO buffer will overflow 4000/X seconds
  from the time the logging was suspended.
  *****)

```

```

USES CRT,
    std_cons,          { unit containing constants used }
    menuinit;          { unit containing initialising procedures }

    { The unit 'std_cons.TPU' consists of the following constants names
      and their corresponding value :

          Setbaud   = 1
          Inselect  = 2
          Fifostat  = 3
          Fifostat  = 4
          Readfifo  = 5
          Rset      = 6
          Clrfullf  = 7
          Control   = 8
      }

CONST
    cardbadd = $310;      { base address of the card : as set up on
                           the DIP switch on the card }

VAR
    filename : STRING[16];
    f1       : FILE OF BYTE;
    datacount : LONGINT;
    databyte  : BYTE;
    status    : BYTE;
    wdbyte    : BYTE;
    x,y       : BOOLEAN;
    filenum   : INTEGER;
    drive     : CHAR;
    action    : CHAR;
    runx      : STRING[3];
    IOCode    : INTEGER;
    keycom    : CHAR;

{-----}
{--- Procedure to write a message and to prompt for an appropriate ---}
{--- action to be taken when 340k of data has been collected -----}
{-----}

PROCEDURE logged340( VAR keyin : CHAR );

BEGIN
    SOUND(4000);      { audible indicator }
    DELAY(100);
    NOSOUND;
    WINDOW(1,1,80,25);
    TEXTBACKGROUND(0);
    TEXTCOLOR(15);
    CLRSCR;
    GOTOXY(5,5);
    WRITELN(' You have collected 340k bytes of data - ',filename,
            ' is closed.');
```

SRL-0058-TM

```

GOTOXY(5,11);
WRITELN(' If floppy drive is used, insert another disk and continue logging.
');
WINDOW(16,7,64,9);
TEXTBACKGROUND(1);
TEXTCOLOR(7);
CLRSCR;
GOTOXY(1,2);
WRITELN(' Incoming data continues to enter FIFO buffer. ');
WINDOW(17,13,62,20);
REPEAT
  CLRSCR;
  GOTOXY(1,2);
  WRITELN(' Press 'f' to immediately begin new file ');
  WRITELN('           on the same disk drive, ');
  WRITELN('           'q' to quit ');
  WRITELN;
  WRITE('           ==> ');
  READLN(keyin);
  keyin := UPCASE(keyin);
  IF NOT (keyin IN ['F','Q']) THEN
    BEGIN
      CLRSCR;
      GOTOXY(30,2);
      WRITELN('Please enter again. ');
      DELAY(1000)
    END;
  UNTIL (keyin IN ['F','Q']);
WINDOW(1,1,80,25);
TEXTBACKGROUND(0);
CLRSCR
END; { of logged340 }

```

```

{-----}
{-- Procedure to prompt for a disk drive to write the collected data to --}
{-----}

```

```

PROCEDURE prompt_drive( VAR disk_drv : CHAR ) ;

```

```

VAR keycom : CHAR;

```

```

BEGIN
  WINDOW(20,8,61,10);
  TEXTBACKGROUND(1);
  TEXTCOLOR(7);
  REPEAT
    CLRSCR;
    GOTOXY(1,2);
    WRITE(' Enter TARGET drive ID (a,b,c,d,e) => ');
    READLN(disk_drv);
    disk_drv := UPCASE(disk_drv);
    IF NOT (disk_drv IN ['A','B','C','D','E']) THEN
      BEGIN
        CLRSCR;

```

```

        GOTOXY(11,2);
        WRITELN('Please enter again. ');
        DELAY(1000)
    END;
UNTIL (disk_drv IN ['A','B','C','D','E']);

    action := 'a';
    WINDOW(20,13,59,15);
    CLRSCR;
    GOTOXY(3,2);
    WRITELN(' Press a key when ready to begin. ');
    REPEAT UNTIL KEYPRESSED;          { wait until keypressed }
    keycom := READKEY;
    WINDOW(1,1,80,25);
    TEXTBACKGROUND(0);
    TEXTCOLOR(15);
    CLRSCR
END;    { of prompt_drive }

{*****}
{***** MAIN PROGRAM *****}
{*****}

BEGIN
    CLRSCR;
    list_set(wdbyte,cardbadd);    { lists menus for serial data configuration
                                and initialises the dart chip on the serial
                                data card }
                                { wdbyte is the mask for the FIFO data bytes
                                and depends on the data word length }

    TEXTCOLOR(7);    { white }
    filenum := 1;

    prompt_drive(drive);

    { the following REPEAT-UNTIL loop facilitates the logging of more
      than one data file (of 340k) and enables consecutive data files
      to have consecutive names }

    REPEAT
        action := 'a';
        x := TRUE;    { x = first time for buffer overflow }
        y := TRUE;    { y = logging data to a new file }
                    { x,y enable messages to be written to screen only
                      once but continuously displayed until cleared }
        datacount := 0;

        REPEAT
            STR(filenum,runx);
            filename := CONCAT(drive,':RUN',runx,'.DAT'); { assigns a name to the
                                                         data file }

            ASSIGN(f1,filename);
            {SI-}
            REWRITE(f1);
            {SI+}

```

SRL-0058-TM

```

IOCode := IOResult;
IF IOCode <> 0 THEN
  BEGIN
    WINDOW(14,3,66,5);
    TEXTBACKGROUND(1);
    TEXTCOLOR(7);
    CLRSCR;
    GOTOXY(1,2);
    WRITELN(' Drive does not exist or is not ready : Try again. ');
    TEXTCOLOR(15);
    prompt_drive(drive);
  END;
UNTIL IOCode = 0;
CLRSCR;

PORT[cardbadd + fifostart] := 1;    { start the pulses to enable writing
                                     the input data to the FIFO }

{ the first three bytes in the FIFO is unwanted data which had resided in the
  receive buffers of the DART - therefore read and discard }

DELAY(10);
databyte := PORT[cardbadd + readfifo];
databyte := PORT[cardbadd + readfifo];
databyte := PORT[cardbadd + readfifo];

{ the following REPEAT-UNTIL loop writes data from the FIFO to
  the data file until 340k has been collected or a key pressed }

REPEAT
  IF y THEN                                { display message if logging data to a new
                                           file or if resumed logging to a file }
  BEGIN
    y := FALSE;
    WINDOW(13,2,65,10);
    TEXTCOLOR(7);    { white (/high) }
    TEXTBACKGROUND(1); { blue (/dim) }
    CLRSCR;
    GOTOXY(1,2);
    WRITELN(' The buffercard is now running... press => ');
    WRITELN;
    WRITELN('    'q' : to close current file and quit ');
    WRITELN('    'n' : to close current file and continue ');
    WRITELN('    'r' : storing incoming data to next file ');
    WRITELN('    'r' : to close current file and ignore ');
    WRITELN('    'r' : incoming data until ready ');
    WINDOW(1,1,80,25);
    TEXTBACKGROUND(0);
    GOTOXY(10,12);
    WRITELN('Logging is suspended when 340k of data has been collected. ');
    GOTOXY(10,14);
    WRITELN('Input data is loaded into file => ',filename)
  END;

```

```

status := PORT(cardbadd + fifostat);      { reads status of the FIFO }

IF (status AND 2) = 2 THEN                  { if D1 is high ie. if FIFO
                                           contains data }
BEGIN
  IF ((status AND 4) = 4) AND x THEN        { if D2 high ie. FIFO
                                           has overflowed }
  BEGIN
    SOUND(5000);                          { audible alarm }
    DELAY(100);
    NOSOUND;
    x := FALSE;
    WINDOW(20,20,59,22);
    TEXTBACKGROUND(1);
    CLRSCR;
    GOTOXY(1,2);
    WRITELN(' Buffer overflow - data has been lost');
    WRITELN(1,1,80,25);
    TEXTBACKGROUND(0)
  END;

  databyte := PORT(cardbadd + readfifo);   { read byte from FIFO }
  databyte := databyte AND wdbyte;          { mask for word length }
  WRITE(f1,databyte);                      { write to file }
  datacount := datacount + 1               { increment count }
END

ELSE
BEGIN
  GOTOXY(30,17);
  WRITELN(datacount, ' bytes')             { display no. of bytes collected}
END;

IF datacount >= 340000 THEN
BEGIN
  CLOSE(f1);
  logged340(action);
  filenum := filenum + 1
END;

IF KEYPRESSED THEN
BEGIN
  keycom := READKEY;                      { read the pressed key }
  keycom := UPCASE(keycom);
  IF keycom IN ['Q','R','N'] THEN
  BEGIN
    WINDOW(1,1,80,25);
    TEXTBACKGROUND(0);
    CLRSCR
  END;
  CASE keycom OF
    'Q' : BEGIN
      CLOSE(f1);
      action := 'Q'
    END;
  END;

```

SRL-0058-TM

```

      'N' : BEGIN
        PORT[cardbadd + clrfullf] := 0;  { clear overflow
                                           flag}
        filenum := filenum + 1;          { increment file number
                                           for next file }
        CLOSE(f1);
        WINDOW(9,2,71,4);
        TEXTBACKGROUND(1);
        CLRSCR;
        GOTOXY(1,2);
        WRITE(' The next 4k bytes of incoming data');
        WRITELN(' is stored in FIFO buffer');
        prompt_drive(drive);
        action := 'N'
      END;
      'R' : BEGIN
        CLOSE(f1);
        PORT[cardbadd + clrfullf] := 0;  { clear overflow
                                           flag}
        filenum := filenum + 1;          { increment file number for
                                           next file }
        PORT[cardbadd + fifostart] := 0;  { inhibit writing
                                           incoming data to the FIFO }
        WINDOW(16,2,64,4);
        TEXTBACKGROUND(1);
        CLRSCR;
        GOTOXY(2,2);
        WRITELN(' Incoming data ignored until logging
                                                         restarted');
        prompt_drive(drive);
        action := 'R'
      END;

    END;
  END;

  UNTIL (action IN ['F','N','R','Q']);

  UNTIL (action = 'Q');
  WINDOW(15,6,65,10);
  TEXTCOLOR(7);
  TEXTBACKGROUND(1);
  CLRSCR;
  GOTOXY(2,2);
  WRITELN(' Logging complete. ');
  WRITELN;
  WRITELN(' ',datacount,' bytes collected in file => ',filename);
  WINDOW(1,1,80,25);
  TEXTCOLOR(15);
  TEXTBACKGROUND(0);

END.

```

TABLE 1

ADDRESSING OFFSETS AND THE FUNCTIONS PERFORMED

The following functions are executed on the card by addressing, in software, the location [base address + offset].

The base address is set by the DIP switch on the card, and the offset is shown below.

A3	A2	A1	A0	OFFSET	FUNCTION
0	0	0	0	0	Transfers a data byte from the PC to the DART (which is then transmitted from the card in a serial form).
0	0	0	1	1	Enables a coded data byte to select the transmit and receive baud rates.
0	0	1	1	2	Enables selection of the RS232 or RS422 interface.
0	0	1	1	3	Reads the status of the FIFO (ie. reads the full and empty flags).
0	1	0	0	4	Enables/disables the transfer of incoming data to the FIFO by enabling/disabling the Dual Retriggerable Monostable module via the start latch.
0	1	0	1	5	Enables the FIFO to output the following data byte onto the bus to be read.
0	1	1	0	6	Resets the FIFO, FIFO overflow latch, DART, and start latch.
0	1	1	1	7	Resets the FIFO overflow latch (only).
1	0	0	0	8	Transfers the control bytes to the DART write registers for initialisation. Also allows the reading of the DART read registers (used in the process of writing bytes out of the card).

- NOTES : 1. ALL ACTIVE INTEGRATED PACKAGES ARE PREFIXED N.
2. X1 IS A 62 WAY CONNECTOR WITH 26 UNUSED PINS.
X3 IS A 9 WAY CANNON D CONNECTOR WITH 1 UNUSED PIN.
X4 IS A 25 WAY CANNON D CONNECTOR WITH 19 UNUSED PINS.
3. SWITCH SETTINGS - S1
- | ADDRESS | A8 | A7 | A6 | A5 | A4 |
|---------|-----|----|----|----|----|
| DIP SW | 1 | 2 | 3 | 4 | 5 |
| HEX | 300 | 00 | 0 | C | C |
| | 310 | 00 | 0 | C | C |
- IF SWITCH 1 CLOSED "C" = Deselect CARD

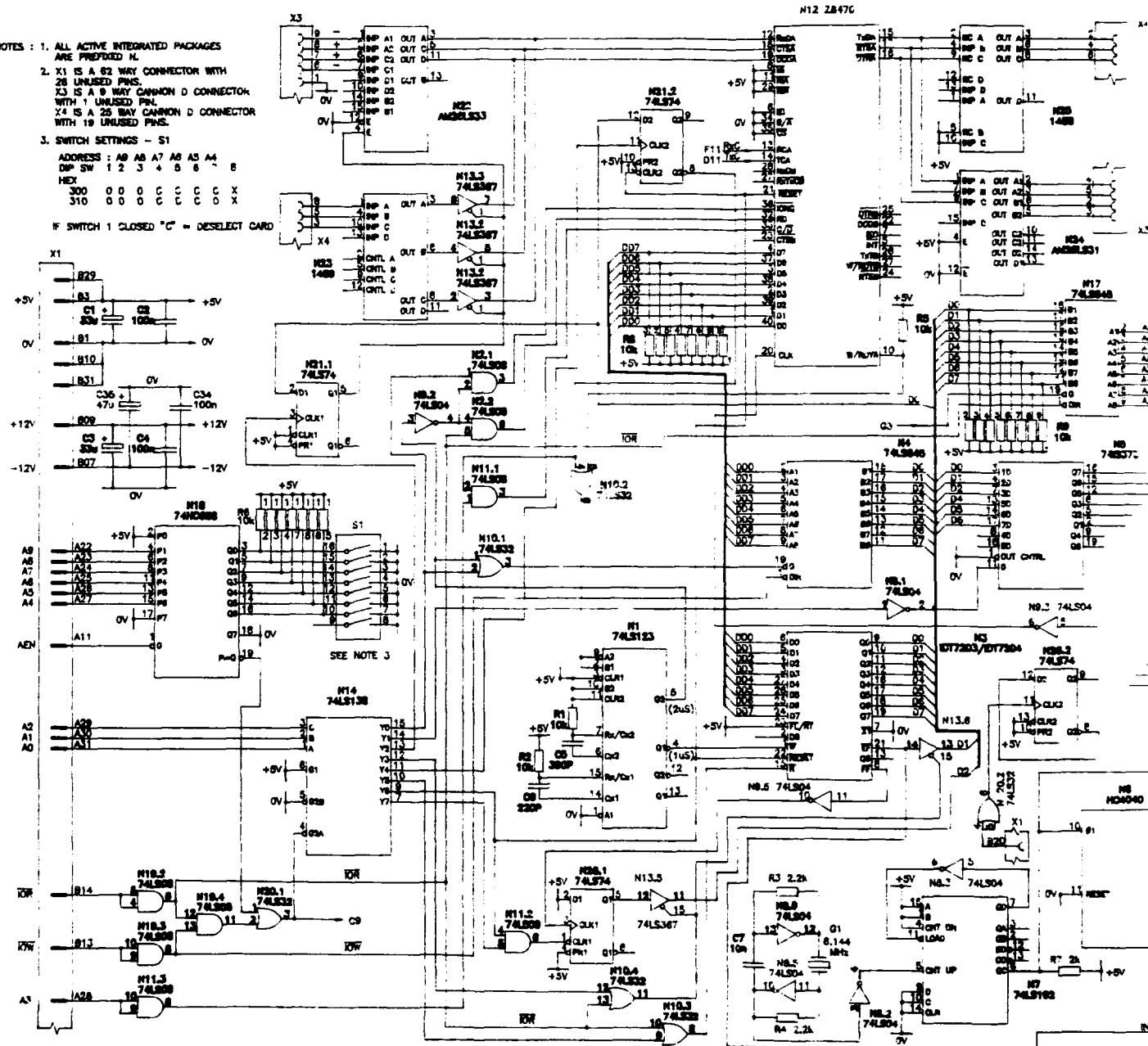
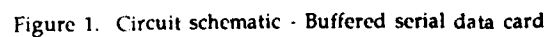
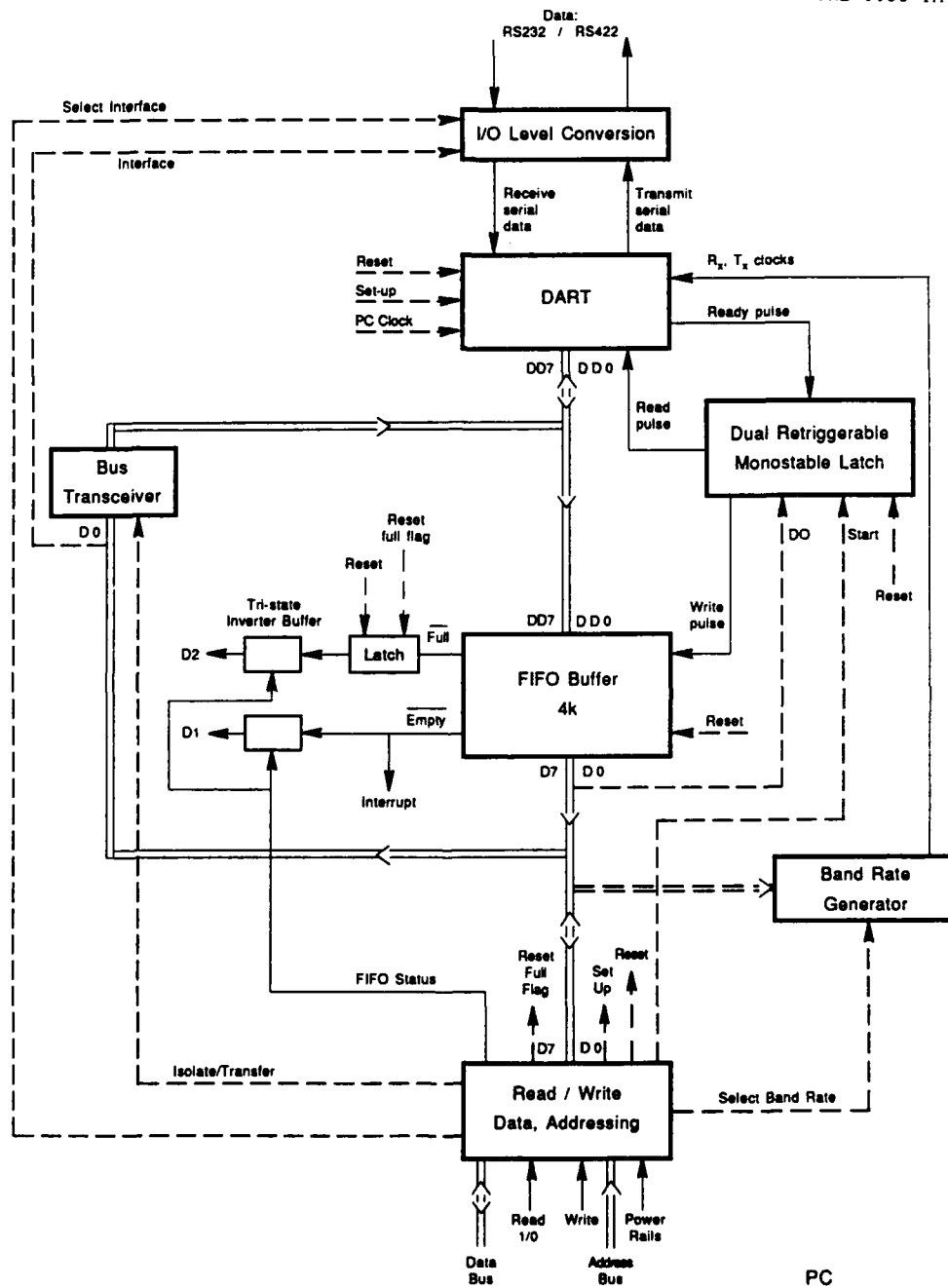


Figure 1. Ci





Note: Dotted lines indicate actual connections but used only during initialisation

Figure 2. Functional units - block diagram

SRL-0058-TM

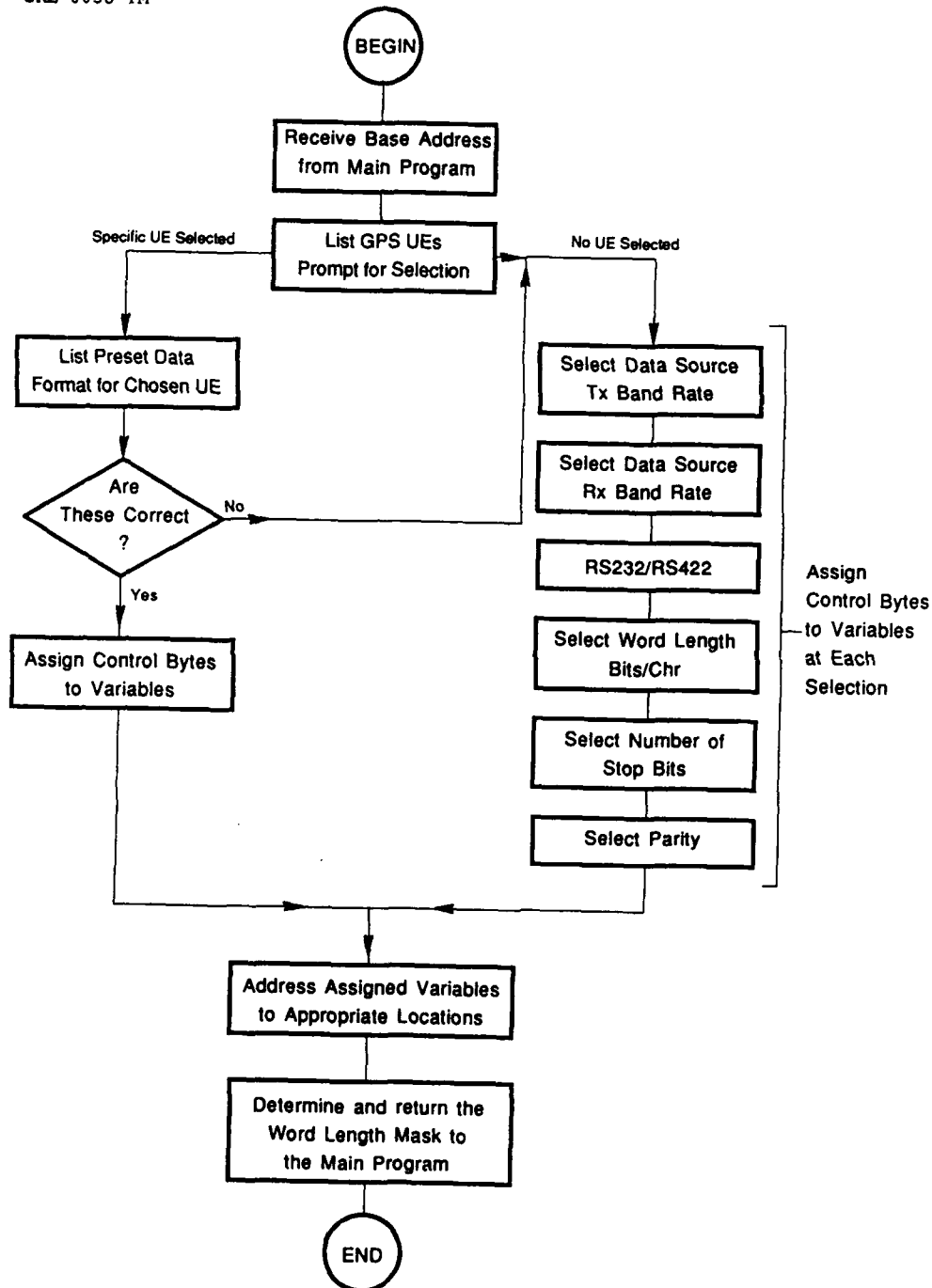


Figure 3. Flowchart of initialisation procedure - used in 'menu init. tpu'

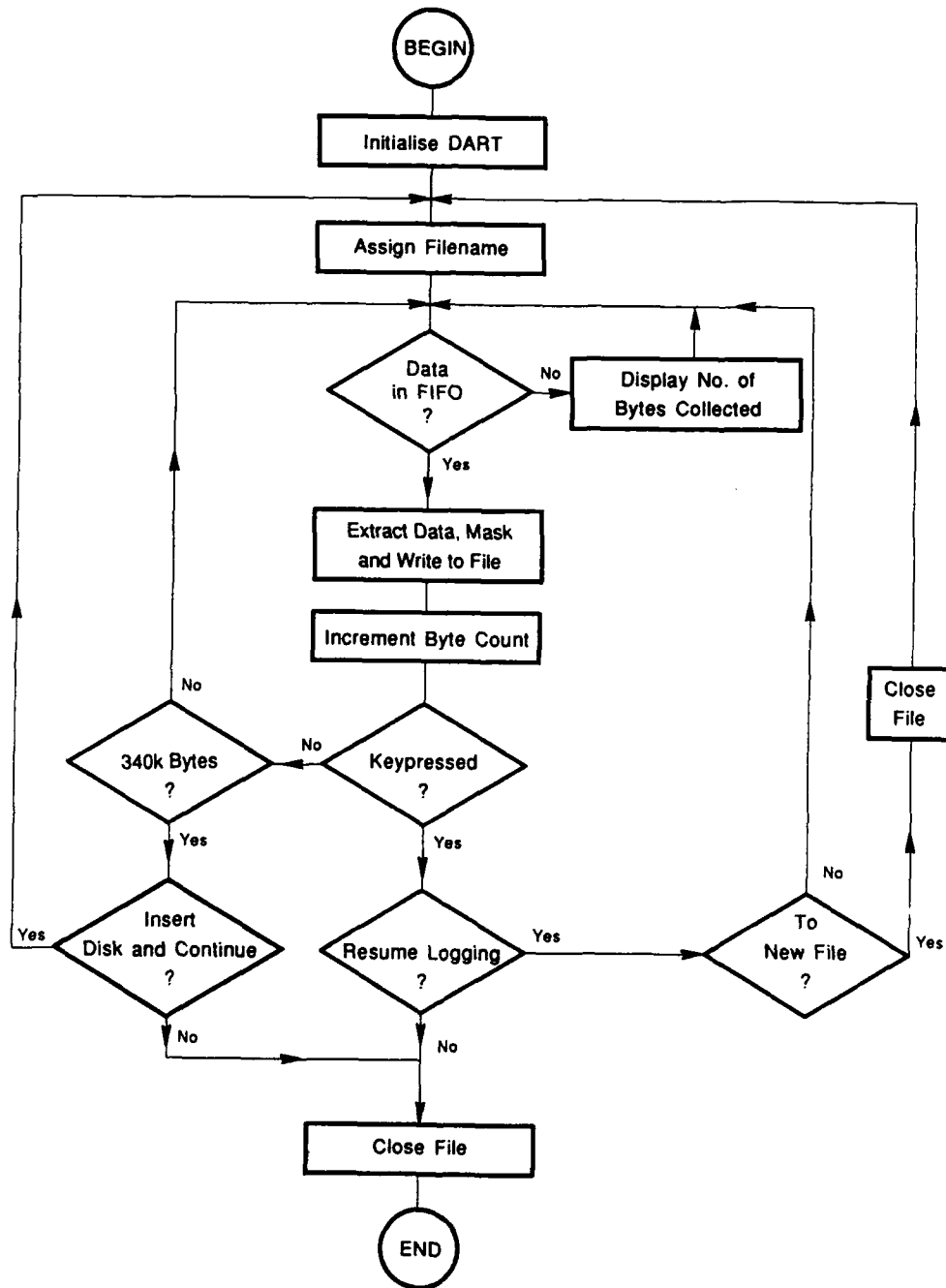


Figure 4. Flowchart of an operating method - used in sample main program

DISTRIBUTION

Copy No.

DEPARTMENT OF DEFENCE

Defence Science and Technology Organisation

Chief Defence Scientist

DSTO Central Office Executive 1

Counsellor, Defence Science, London Cnt Sht Only

Counsellor, Defence Science, Washington Cnt Sht Only

Navy Scientific Adviser (NSA) 2

Scientific Adviser - Army (SA-A) 3

Air Force Scientific Adviser (AFSA) 4

Scientific Adviser - Defence Central (SA-DC) 5

Aeronautical Research Laboratory

Dr. Robbin Miller 6

Library, ARL 7

Electronics Research Laboratory

Director, Electronics Research Laboratory 8

Surveillance Research Laboratory

Director, Surveillance Research Laboratory 9

Chief, Microwave Radar Division 10

Head, Microwave Radar Engineering 11

Mr. J Silby, Microwave Radar Engineering 12

Mr. A Padgham, Microwave Radar Engineering 13

Mr. D Ireland, Microwave Radar Engineering 14

Weapons Systems Research Laboratory

Dr. Dan Keenan 15

Joint Intelligence Organisation (DSTI) 16

Director of Departmental Publications 17

SRL-0058-TM

RANTAU	18
SA-DIO	19
Libraries and Information Services	
Librarian, Technical Reports Centre, Defence Central Library, Campbell Park	20
Manager Document Exchange Centre	21
US National Technical Information Service	22,23
UK Defence Research Information Centre	24,25
Director Scientific Information Services (Canada)	26
NZ Ministry of Defence	27
National Library of Australia	28
Main Library, Defence Science and Technology Organisation Salisbury	29,30
Library, MRL	31
Librarian, DSD, Melbourne	32
Australian Defence Force Academy Library	33
British Library Document Supply Centre	34
Director Publications for AGPS	35
Author	36
Spares	37,38

DOCUMENT CONTROL DATA SHEET

Security classification of this page :

UNCLASSIFIED

1 DOCUMENT NUMBERS AR Number : AR-006-440 Series Number : SRL-0058-TM Other Numbers :	2 SECURITY CLASSIFICATION a. Complete Document : Unclassified b. Title in Isolation : Unclassified c. Summary in Isolation : Unclassified 3 DOWNGRADING / DELIMITING INSTRUCTIONS
4 TITLE BUFFERED SERIAL DATA CARD	
5 PERSONAL AUTHOR (S) G. Fielke	6 DOCUMENT DATE May 1991 7 7.1 TOTAL NUMBER OF PAGES 27 7.2 NUMBER OF REFERENCES 2
8 8.1 CORPORATE AUTHOR (S) Surveillance Research Laboratory 8.2 DOCUMENT SERIES and NUMBER Technical Memorandum 0058	9 REFERENCE NUMBERS a. Task : Air 89/215 b. Sponsoring Agency : 10 COST CODE
11 IMPRINT (Publishing organisation) Defence Science and Technology Organisation	12 COMPUTER PROGRAM (S) (Title (s) and language (s))
13 RELEASE LIMITATIONS (of the document) Distribution: Approved for public release	

Security classification of this page :

UNCLASSIFIED

Security classification of this page :

UNCLASSIFIED

14 ANNOUNCEMENT LIMITATIONS (of the information on these pages)

No limitations

15 DESCRIPTORS

- | | |
|-----------------------------|---------------------------|
| a. EJC Thesaurus
Terms | Global positioning system |
| b. Non - Thesaurus
Terms | Buffered serial data card |

16 COSATI CODES

170705
1205

17 SUMMARY OR ABSTRACT

(if this is security classified, the announcement of this report will be similarly classified)

NAVSTAR Global Positioning System (GPS) User Equipments (UEs) send out navigation information in the form of blocks of serial data. The format and transmitting rate of the data varies between UEs.

Several agencies within the Department of Defence require the collection of GPS UE data.

As a personal computer general purpose data logging card was not available commercially, it was decided to design and develop such a card to enable collection of data from any GPS UE.

The card may be used to collect data from any source which transmits data at either RS232 or RS422 levels.

This manual describes the operation of the Buffered Serial Data Card and the process of writing user software.

Security classification of this page :

UNCLASSIFIED